Implementation of Multispectral Image Classification on a Remote Adaptive Computer

Marco A. Figueiredo [a], Clay S. Gloster [b], Mark Stephens [c],
Corey A. Graves[b], Mouna Nakkar[b]

[a] SGT Inc., Code 564, NASA Goddard Space Flight Center,
Greenbelt Road, Greenbelt, Maryland, 20771
[b]Electrical and Computer Engineering, North Carolina State University
Box 7914, NCSU, Raleigh NC 27695-7914
[c] Code 585, NASA Goddard Space Flight Center,
Greenbelt Road, Greenbelt, Maryland, 20771

(*Please send all correspondence to Clay Gloster*)
Electrical and Computer Engineering, North Carolina State University
Box 7914, NCSU, Raleigh NC 27695-7914
E-mail: gloster@eos.ncsu.edu
phone: (919) 515-7348
fax: (919) 515-2285

E-mail addresses of the other authors
Marco Figueiredo: marco@fpga.gsfc.nasa.gov
Mouna Nakkar : mouna@ibmoto.com
Mark Stephens: Mark.A.Stephen.1@gsfc.nasa.gov
Corey Graves: cagrave@eos.ncsu.edu

Abstract

As the demand for higher performance computers for the processing of remote sensing science algorithms increases, the need to investigate new computing paradigms is justified. Field Programmable Gate Arrays enable the implementation of algorithms at the hardware gate level, leading to orders of magnitude performance increase over microprocessor based systems. The automatic classification of spaceborne multispectral images is an example of a computation intensive application that can benefit from implementation on an FPGA-based custom computing machine (adaptive or reconfigurable computer). A probabilistic neural network is used here to classify pixels of a multispectral LANDSAT-2 image. The implementation described utilizes Java client/server application programs to access the adaptive computer from a remote site. Results verify that a remote hardware version of the algorithm (implemented on an adaptive computer) is significantly faster than a local software version of the same algorithm (implemented on a typical general-purpose computer).

I. INTRODUCTION

A new generation of satellites is being developed by the National Aeronautics and Space Administration (NASA) to compose the Earth Observing System (EOS). The instruments aboard the EOS satellites not only extend the observation life of the current satellites, but they also extend the capabilities of remote sensing scientists to better understand the Earth's environment. Along with the scientific advancements of the new missions, it is also necessary to explore new technologies that facilitate and reduce the cost of the data analysis process. In order to process the high volume of data generated by the new EOS satellites, NASA is constructing the Distributed Active Archive Centers (DAACs), an extensive and powerful parallel computing environment. Scientists will be able to request certain data products from these centers for further analysis on their own computing systems. A new technology that could bring increased processing power to the scientist's desk, offering more complex analysis and interpretation of remote sensed scientific data, is highly desirable. The ultimate scenario would be for the scientist to request the data directly from the satellite along with historic data from an archive center.

Field Programmable Gate Array (FPGA)-based computing, also known as "adaptive" or "reconfigurable computing", has emerged as a viable computing option in computationally intensive applications. These computing systems combine the flexibility of general purpose processors with the speed of application specific processors. By mapping hardware to FPGAs, the computer designer can optimize the hardware for a specific application resulting in acceleration rates of several orders of magnitude over general purpose computers. Because the FPGAs are personalized using SRAM-based memory cells or a fuse programming technology, they can be reconfigured by the designer for other applications.

Several reconfigurable computers have been implemented to demonstrate the viability of reconfigurable processors [1], [2], [3], [4]. Applications mapped to these processors include: pattern recognition in high-energy physics [5], applications in statistical physics[6], and genetic optimization algorithms [7], [8]. In many cases [9], [10], [11], the reconfigurable computing implementation provided the highest

performance, in terms of execution speed. The advent of reconfigurable processors along with novel methods for mapping applications onto adaptive or reconfigurable processors enables a new computing paradigm that may represent the future for remote sensing scientific data processing. In fact, many applications utilizing FPGA based computers have been developed showing orders of magnitude acceleration over microprocessor based systems [12],[13],[14]. Moreover, microprocessors and FPGAs share the same underlying technology - the silicon fabrication process. Therefore, it is reasonable to conclude that FPGA based machines can usually outperform microprocessor based systems by orders of magnitude [15], [16],[17].

To achieve such performance, the application must effectively utilize the available resources. This presents a challenge for software designers, who are generally accustomed to mapping applications onto fixed computing systems. Generally, the designers examine the available hardware resources, then modify their application accordingly. With reconfigurable computers, the available resources can be generated as needed. While it may seem that this flexibility would ease the mapping process, it actually introduces new problems, such as what components should be used, and how many of each component should be used to generate the best performance. With conventional hardware components, these questions are less of an issue. In addition, software engineers are generally not adept at hardware design. Thus, several research groups have developed methods for mapping applications to reconfigurable processors [2], [18], [19], [20], [21].

The Adaptive Scientific Data Processing (ASDP) group at NASA's Goddard Space Flight Center (GSFC), in conjunction with researchers at North Carolina State University, have been investigating the utilization of FPGA-based computing in the processing of remote sensing scientific algorithms. The first prototype developed by the group utilized a commercial-off-the-shelf (COTS) reconfigurable accelerator in the implementation of an automatic classifier for the LANDSAT-2 multispectral images [22]. The implementation discussed in this paper is an extension of the original prototype that allows users to classify the images on the accelerator from a remote site. Results indicate that a remote implementation of the classifier in adaptive computing hardware is faster than a software implementation that executes on a local high-end workstation.

This paper presents details of the FPGA design and is organized as follows. Section 2 describes the classifier algorithm that utilizes a probabilistic neural network (PNN). The implementation of the FPGA custom computing machine is then presented. Finally, a performance analysis of local and remote versions of the algorithm is presented.

II. The PNN multispectral image classifier

Remote sensing satellites utilize multispectral scanners to collect information about the Earth's environment [23]. The data collected by such instruments are a set of images, each corresponding to one spectral band. A multispectral image pixel is represented by a vector of size equal to the number of bands. The combination of the multiple spectrum measurements represented by each element of the pixel vector determine a signature that corresponds to a physical object being viewed by the satellite. Through the observation of a multispectral image and the comparison of pixel vectors to those obtained from known locations (in-situ measurements), a scientist is able to identify unique signatures of physical objects and compose classes. These classes contain multispectral pixel representations of physical objects on the earth that are closely related. Example classes include forest, tundra, wetland, water, etc.

Several neural network schemes have been devised for the automatic classification of multispectral images [24]. One in particular, the Probabilistic Neural Network (PNN) classifier [25], exhibits acceptable accuracy, very small training time, robustness to weight changes, and negligible retraining time. A description of the derivation of the PNN classifier and details of the network implementation including rate of false alarms, neural network size, etc. are presented in *Chettri et. al.* [25]. The Blackhills (South Dakota, USA) data set was generated by the Landsat 2 multispectral scanner (MSS). The image's four spectral bands (0.5-0.6 μm, 0.6-0.7 μm, 0.7-0.8 μm, and 0.8-1.1 μm) correspond to channels 4 through 7 of the Landsat MSS sensor. There are 262,144 pixels corresponding to a 512x5l2 pixel image size, and each pixel represents a 76m x 76m ground area; the images were obtained in 1973. The ground truth was provided by the United States Geological Survey.

Figure 1 illustrates the PNN classifier procedure. Each multispectral pixel, represented by a vector, is compared to a set of pixels belonging to a class. A probability value is calculated for each class. The highest value indicates the class into which the pixel fits. Eq. 1 is used to derive a value that indicates the probability that the pixel fits into class S_k.

$$f(X \mid S_k) = K1[k] \sum_{i=1}^{P_k} e^{[-K2[k](\vec{X}-\vec{W}_{ki})^T(\vec{X}-\vec{W}_{ki})]} \tag{1}$$

where ($\vec{X}$ is a pixel vector, $\vec{W}^{ki}$ is the weight i of class k, d is the number of bands, k is the number of classes, P_k is the number of weights per class, and $K1[k]$, $K2[k]$ are constants.)

III. The FPGA implementation

The first step in implementing an application on an adaptive computer is to select the FPGA-based custom coprocessor architecture that best matches the algorithm in question. At the current state of the technology, certain FPGA architectures provide better performance than others for a particular class of applications. A preliminary analysis of the PNN classifier indicated that the FPGA architecture [26], shown in Figure 2, matched well with the algorithm. The selected FPGA architecture is composed of a PCI bus based motherboard and up to 16 plug-in modules. These plug-in modules each contain two Xilinx 4013E FPGA devices(XFPGA and YFPGA) and provide an equivalent of 13,000 gates per FPGA, or 26,000 gates per module. The design implementation required approximately 1160 CLBs (85% utilization) per FPGA. Since the module contained two FPGAs and two separate memory modules (connected via the HBUS), we can perform two lookup table (LUT) operations simultaneously.

A. Algorithm partitioning

The computation intensive portion of the multispectral image classification algorithm found in Eq. 1 was identified by profiling an implementation of the algorithm that was written using the C programming language. This computation was selected to be executed on the FPGA coprocessor to improve performance for the complete classification algorithm. The graphical user interface, data storage, adaptive coprocessor initialization code, algorithm synchronization, and data I/O is performed by the host processor. The compute intensive PNN classification algorithm equations were mapped onto a single module.

Figure 3 illustrates the algorithm partitioning. The host processor displays the image during classification. The host then sends a pixel vector to the FPGA coprocessor. Classification is performed on the coprocessor and results are returned to the host to be displayed. The host also computes the total time required to process a complete image. If we wish to use multiple modules as coprocessors, the host schedules a pixel vector to be processed on each module in a round-robin fashion, then gathers the results as they become available.

B. FPGA application design

Due to the limited number of gates available on a single FPGA, it was not feasible to use floating point arithmetic in our implementation of the PNN algorithm. We therefore transformed the algorithm to use fixed point arithmetic prior to hardware implementation. The width of the fixed point datapath was determined by simulating variable bit operations in C and comparing the results obtained from

the original algorithm in floating point. Once the fixed point classification of the Blackhills data set yielded exactly the same results as the floating point version, the data path width for the FPGA implementation was known. (Since the output of the PNN classifier is simply a 4-bit value representing the class that matches the pixel, the fixed point version produced exactly the same result as the floating point version. Hence there is no loss in precision due to implementation using fixed point arithmetic.)

Figure 4 shows the data flow diagram for the hardware implementation of the PNN classifier. A portion of the design was mapped onto the XFPGA and the remaining blocks were implemented on the YFPGA of the module. The number of bands (d) was fixed to 4, the maximum value of the number of weights per class (P_k) was fixed to 512, and the maximum number of classes (k) was set to 16. As shown in Eq 1, there are two constants, $K1$ and $K2$, that are class dependent. These constants are pre-calculated on the host and downloaded to memory banks residing on the modules.

The weight memory was mapped to the SRAM that is connected to the YFPGA on the module. The weight memory can be as large as 16*512*4*2 bytes = 32768 16-bit words. Each weight value occupies 10-bits. Since each class can have up to 512 weights, an array that holds the number of weights for each class is employed. The inputs of the array are also visible from the host processor.

A 4-bit register holds the number of classes. This register is initialized by the host before loading the FPGA coprocessors. Due to the lack of space on the XFPGA, the $K1$ multiplier and the class comparison blocks were moved to the host. These calculations amount to k multiplications and comparisons per pixel classification. Since the number of classes, k, is small, they do not account for a significant amount of the computation, leading to a small performance penalty. For example, if the number of classes $k = 16$, the maximum number of weights per class $P_k = 512$, and we are classifying a 512x512 image with $d = 4$ spectral bands, Eq. 1 is calculated 16 times. The performance penalty amounts to only 16 multiplications and 16 comparisons per 512x512 image that are executed on the host rather than executed on the FPGA. This is a small overhead relative to the more than 512^3 multiplications that are computed on the FPGA for this example.

Figure 4 contains a Subtraction Unit that computes W, a 4 x 10-bit element vector for W (w_0, w_1, w_2, w_3) minus X (x_0, x_1, x_2, x_3). The result of the subtraction ranges from -1023 to 1023, requiring 11 bits in two's complement format. The Square Unit multiplies each 11-bit element of the Y vector by itself (i.e. $t_0 = y_0 \times y_0$). The values of the elements of the T vector range from 0 to 1,046,529, requiring 20 bits in two's complement format.

The next computation involves the Band Accumulator Unit. This unit adds the 4 elements of the T vector together resulting in u, ranging in value from 0 to 4,186,116, requiring 22 bits. The $K2[k]$

Memory holds the $K2$ values for each class. $K2 = (1/2)\sigma_k^-2$, where $\sigma_k = 2, 3, \ldots, 11, 12$. As a result, $K2$ varies between 0.125 ($\sigma_k = 2$), and 0.003472 ($\sigma_k = 12$). The largest value of $K2 = 0.125$ in decimal and is represented exactly in binary (0.001). In order to increase the precision of the multiplication, the values of $K2$ are stored with the decimal point shifted to the right by 2 (multiplied by four). After $K2$ is multiplied by u in the $K2$ Multiplier Unit, the decimal point of the result of the multiplication is shifted to the left by 2 (divide by 4 effect). Since this is a representation issue, no hardware is necessary to perform the shifts in the YFPGA (refer to Figure 4), only the host needs to maintain the values in the $K2[k]$ memory in the appropriate format. The $K2$ Multiplier Unit multiplies the $K2$ values for each class by the accumulated values of the difference between a pixel and a weight vector. It delivers a 44-bit result to the TO_XFPGA unit shown in Figure 4. Bits 0 to 23 represent the fraction portion (remember that the decimal point is shifted to the left by 2), and bits 24 to 43 represent the integer part of the result.

The next operation is to compute the exponential of the negative of this number. Given the precision of the following operations, any number above 24 will yield zero as a result. Thus, if any of bits 43 to 29 is set or both bits 28 and 27 are set, the result of e^{-a} should be zero. Only 28 bits are passed on to the Exponential LUT Unit, and they are bits 1 to 28. Bit 0 and bits 29 to 43 are discarded. It was also found that a considerable number of results of the multiplication are zero, which indicates that the result of the exponential should be one. In order to save processing steps in this case, the output of the multiplier is tested for zero, and a flag is passed to the Exponential LUT Unit, indicating that its result should be 1.

A look-up table is used to determine the value of e^{-a}. If we assume that $a = b + c$, then:

$$e^{-a} = e^{-(b+c)} = e^{-b}e^{-c} \tag{2}$$

Since a is a 28-bit binary number, the value comprising bits 27 to 14 of a represent b, and the value comprising bits 13 to 0 of a represent c. The range of values of b and e^{-b} are:

$$00000.000000000 \leq b \leq 10111.111111111, \tag{3}$$

or

$$0 \leq b \leq 23.9980469, \tag{4}$$

which results in

$$0.9980519 \geq e^{-b} \geq 3.78 \times 10^{-11} \tag{5}$$

The range of values of c and e^{-c} are:

$$00000.00000000000000000000001 \leq c \leq 00000.00000000011111111111111, \tag{6}$$

or

$$1.19 \times 10^{-7} \leq c \leq 1.8919 \times 10^{-3}, \tag{7}$$

which results in

$$0.999999881 \geq e^{-c} \geq 0.998109888 \tag{8}$$

The values of e^{-b} and e^{-c} are previously calculated and organized into a look-up table. At run time, the values of b and c are used to address the look-up table stored in the memory that is directly connected to the XFPGA. The values of e^{-b} and e^{-c} retrieved from the look-up table are then multiplied to give the value of e^{-b}. The values stored in the look-up table are 32-bits wide. The result of the multiplication is 64-bits, but only the most significant 32 bits are sent out. As a result,

$$3.77 \times 10^{-11} \leq e^{-a} \leq 0.998051781. \tag{9}$$

The Class Accumulator Unit sums up all the comparisons between a given pixel and all weights of a given class, and outputs the result when it receives a flag indicating that the data to add to the accumulator refers to the last weight in a class. The output of the Exponential Multiplier Unit range is $3.77.10^{-11} \leq d \leq 0.998051781$. Thus, the largest accumulated value is 0.998051781 * 512 (max. of weights) = 511.002511872. In order to keep the precision of d, the accumulator is extended to 40 bits to accommodate the original 31 bits after the decimal point and 1 bit before the decimal point, and the new 8 bits before the decimal point. Each class has a $K1$ value associated with it. The value of $K1$ is determined by the following formula:

$$K1 = \frac{1}{(2\pi)^{d/2}\sigma_k^d P_k} \tag{10}$$

The result of the multiplication of $K1$ by the accumulated differences between a pixel and all weights in a given class is compared with all other classes to determine the largest result, which indicates in which class a pixel most probably belongs. In order to keep the values being multiplied in the same range allowing us to use fixed point arithmetic, the values of Kl are normalized as follows: Given d, σ_k and P_k, the host program calculates all $K1$ values, and divides them by the largest one. The result is that one value of $K1$ equals 1 and all the others are less than 1. The $K1$ Multiplier Unit multiplies

the 40-bit result of the Class Accumulator Unit by the 32-bit $K1$ value from the $K1$ Memory Unit, and outputs a 40-bit result to the g register in the Class Comparison Unit. The Class Comparison Unit receives a value that represents the comparison between a pixel and all weights in a class, and compares this value against the values generated for all other classes. At the end of the calculation of all classes, it outputs a code that represents the class which presented the largest value or is the closest match.

C. The host software

The software that was developed for the PNN algorithm that executes on the host processor was written in the Java programming language. We selected the Java programming language for several reasons. Java supports software reuse, native methods, remote method invocation, and it has a built-in security manager. Software reuse allows Java objects and methods to be used repeatedly in different applications. Native methods allow legacy code (old software written in another language) to be called directly from Java methods. The security manager and remote method invocation allow Java programs to be executed on remote CPUs with the system taking care of network traffic errors, security, etc. The FPGA system used for development of the hardware modules, contains drivers for interfacing to the FPGA devices that are only available in the C programming language. Java, was a useful choice for a programming language since native methods allow one to call C routines directly from Java. This is accomplished by building a dynamic link library that contains the C functions that interface to the FPGA coprocessors. A Java native method is used to call these C functions directly.

The application was implemented using a client/server methodology to provide an interface to the FPGA coprocessors from a remote site. The server program interfaces directly to the reconfigurable accelerator via the C drivers. It receives a block of pixels from the client, initiates the classification of each of the pixels on the FPGA accelerator, gathers the results into a block of classified data, and sends the results back to the client. The client software controls the user interface, image data input/output and translation, in addition to communication with the server. By selecting Java as a programming language and separating the program into client and server subsystems, the client software is completely independent of the operating system that will execute the client program. Only the server contains code that is not only dependent on the operating system used, but also depends on the specific reconfigurable accelerator that has been selected. Hence, in this paper we present results obtained from an implementation of the PNN algorithm that can be executed from a remote machine accessible, for example, on the Internet.

IV. Experimental Results

In our experiments, we used the remote implementation of the PNN classifier to measure the effectiveness of a client/server approach to adaptive computing. Figure 5 illustrates a potential scenario for remote image classification. In this configuration, the server program has a direct interface to the FPGA coprocessor. It initializes the FPGA board and loads the architecture shown in Figure 4 into the programmable hardware. In our project, the server executes on a workstation at NASA. The client program communicates with the server via the Internet. The client requests a connection with the server and, once granted, sends data to the server for processing. The server processes the data and sends the results back to the client for display. While the client is designed to execute at a remote site, e.g. NCSU, in our experiments, both the client and server programs were executed on a single host at NASA.

Two software implementations of the PNN algorithm were developed to compare the relative performance of implementations in two different programming languages. One version was written entirely in the Java programming language. The other version was written using the C programming language. The main routine in the client spawned either the Java or C versions of the algorithm via a call from a normal or native method respectively.

Two FPGA-based hardware versions of the PNN algorithm were implemented using single or multiple modules. We report results using two modules as we only had two modules available for our experiments. In the single module case, one pixel or one block of pixels were sent to each FPGA coprocessor and the results were returned to the client via the server. In the two module experiments, one pixel or one block was sent to each of the two FPGA coprocessors in an attempt to speedup algorithm execution by a factor of 2. Each module in the multiple module case contained a complete implementation of the hardware in Figure 4.

A traditional version of the PNN Classifier algorithm was previously developed as the basis for the remote version presented in this paper. This experiment demonstrates the potential merits of a remote image classification algorithm implementation. The traditional version executed on a 100 MHz Pentium PC. This implementation, written entirely in C, required 2043 CPU seconds to classify the complete Blackhills data set. By augmenting the PC with a single module running the PNN classifier at 16 MHz, the processing time was reduced to 220 CPU seconds. In this case, the adaptive computing implementation is 9.29 times faster than the software version. Adding one additional module improved execution time to 90 CPU seconds.

In our experiments with the remote PNN classifier, we ran a total of 4 different scenarios presented

in Figure 6. The scenarios allow us to compare local and remote versions of the algorithm that execute on the client and server with pixel-based or block-based algorithms where one pixel or one block of pixels is processed. In each experiment, we present execution times for two software implementations (written in Java and C) and two hardware implementations (one module or two modules).

In Table I, we present results of a remote implementation of the image classification algorithm where one pixel is processed at a time. Note that the implementation of the algorithm in Java requires 7598 CPU seconds to complete. The C version of the algorithm requires slightly more time since it is actually spawned from the local client Java program to execute on the remote server workstation. (The overhead associated with calling a C function from Java is included in the execution time.) For all practical purposes, the C and Java versions of the algorithm require approximately the same execution time. This was a strange result since Java is an interpreted language, however, we noticed a drastic improvement in the execution of Java programs using more recent versions of the Java interpreter. The remote version of the algorithm executing on a single FPGA module was 3.57 times faster than the remote software version. Also note that the addition of one module in the multiple module case does not impact performance.

The next experiment involved sending a block of data from the client to the server for processing. The results of this experiment are also shown in Table I. In our experiments, an arbitrary block size (equal to 6 rows) was selected. (Future experiments will identify the optimal block size.) Since there are 512 pixel vectors in a row, and 4 pixels per vector, one block contains 12,228 pixels. Note that the execution time of the remote Java version of the block-based algorithm is significantly smaller than the pixel-based algorithm. The execution time reduced from 7598 to 1358 CPU seconds. Once again, the single module implementation was significantly (7.6 times) faster than the remote software version written in Java. The addition of a second module did not provide a speedup due to the overhead associated with sending a block of data to the server. Please note that the FPGA coprocessor consistently processes a pixel at a time, however, the server will wait for all pixels in a block to be processed before sending the results back to the client.

Table II presents results of PNN classification executing on a local workstation. The client program can initiate execution of either of the software or hardware algorithm implementations. In the local pixel-based algorithm, the Java version requires about 1317 CPU seconds and the single module implementation requires 141 CPU seconds. This is approximately an order of magnitude improvement in execution time. The multiple module version completes in 77 seconds resulting in a 2:1 speedup over the single module as expected. The results from Table II illustrate that block-based processing is

counterproductive on a local client workstation.

V. Conclusions

In this paper, it was shown that the implementation of a multispectral image classifier on an adaptive computer yields an order of magnitude performance increase over high end workstations. If we extract the fastest execution times for the algorithm from the Tables presented, we find an interesting result that relates to the potential impact of remote adaptive computing technology. The fastest remote hardware implementation of the PNN algorithm consisted of a single module requiring 178 CPU seconds to complete. On the other hand, the fastest local software version of the algorithm was the Java version that required 1309 CPU seconds. This is 7.35 times slower than the remote hardware implementation. Hence, for image classification, a remote hardware implementation of the algorithm is faster than a local software implementation of the algorithm. Future work is to identify additional applications wherein a remote hardware implementation is consistently faster than a local software version. Additionally experiments that quantify the effects of a heavily loaded network connection should be conducted.

References

[1] J. Hess, D. Lee, S. Harper, M. Jones, and P. Athanas, "Implementation and Evaluation of a Prototype Reconfigurable Router," in *Seventh IEEE Workshop on FPGAs for Custom Computing Machines*, 1999.

[2] M. J. Wirthlin and B. L. Hutchings, "DISC: A Dynamic Instruction Set Computer," in *Third IEEE Workshop on FPGAs for Custom Computing Machines*, 1995.

[3] L. Agarwal, M Wazlowski, and S. Ghosh, "An Asynchronous Approach to Efficient Execution of Programs on Adaptive Architectures Utilizing FPGAs," in *Second IEEE Workshop on FPGAs for Custom Computing Machines*, 1994, pp. 101–110.

[4] P. Bertin and Herve' Touati, "PAM Programming Environments: Practice and Experience," in *Second IEEE Workshop on FPGAs for Custom Computing Machines*, 1994, pp. 133–138.

[5] H. Hogl, A. Kugel, J. Ludvig, R. Manner, K. H. Noffz, and R. Zoz, "Enable++: A Second Generation FPGA Processor," in *Third IEEE Workshop on FPGAs for Custom Computing Machines*, 1995.

[6] C. P. Cowen and S. Monaghan, "A Reconfigurable Monte-Carlo Clustering Processor (MCCP)," in *Second IEEE Workshop on FPGAs for Custom Computing Machines*, 1994, pp. 59–65.

[7] S. D. Scott, A. Samal, and S. Seth, "HGA: A Hardware-Based Genetic Algorithm," in *ACM/SIGDA Symposium on Field Programmable Gate Arrays*, 1995, pp. 53–59.

[8] P. Graham and B. Nelson, "Genetic Algorithms in Software and in Hardware," in *Fourth IEEE Workshop on FPGAs for Custom Computing Machines*, 1996.

[9] W. Luk, T. Lee, J. Rice, and P. Cheng, "Reconfigurable Computing Augmented Reality," in *Seventh IEEE Workshop on FPGAs for Custom Computing Machines*, 1999.

[10] M. Gokhale, B. Holmes, A. Kopser, D. Kunze, D. Lopresti, S. Lucas, R. Minnich, and P. Olsen, "Splash: A Reconfigurable Linear Logic Array," in *International Conference on Parallel Processing*, 1990, pp. 526–532.

[11] J. Vuillemin, P. Bertin, D. Roncin, M. Shand, H. Touti, and P. Boucard, "Programmable Active Memories: Reconfigurable Systems Come of Age," *IEEE Transactions on VLSI Systems*, vol. 4, no. 1, pp. 56–69, 1996.

[12] P. Athanas and L. Abbott, "Real-time image processing on a custom computing platform," *IEEE Computer*, vol. 28, pp. 16–24, February 1995.

[13] N. Shirazi P. Athanas and A. Abbott, "Implementation of a 2-D fast fourier transform on an FPGA-based custom computing machine," *Proceedings of the 5th International Workshop on Field-Programmable Logic and Applications*, vol. 1, pp. 282–292, 1995.

[14] B. Hutchings M. Rencher, "Automated Target Recognition on Splash-II," *IEEE Symposium on Field Programmable Custom Computing Machines*, vol. 1, pp. 232–240, April 1997.

[15] K. Bazargan and M. Sarrafzadeh, "Reconfigurable Computing Augmented Reality," in *Seventh IEEE Workshop on FPGAs for Custom Computing Machines*, 1999.

[16] Get ready for reconfigurable computing, ," *Computer Design*, vol. 1, pp. 55–63, April 1998.

[17] T. Hoffmann U. Nageldinger R. Hartenstein M. Her and U Kaiserslautern, "On reconfigurable co-processing units," *Proceedings of the 5th Reconfigurable Architectures Workshop (RAW'98)*, vol. 1, March 30 1998.

[18] M Wazlowski, L. Agarwal, T. Lee, A. Smith, E. Lam, P. Athanas, H. Silverman, and S. Ghosh, "PRISM-II Compiler and Architecture," in *First IEEE Workshop on FPGAs for Custom Computing Machines*, 1993, pp. 9–16.
[19] P. M. Athanas and H. F. Silverman, "Processor Reconfiguration Through Instruction-Set Metamorphosis," *IEEE Computer*, vol. 26, no. 3, pp. 11–18, 1993.
[20] D. Galloway, "The Transmogrifier C Hardware Description Language and Compiler for FPGAs," in *Third IEEE Workshop on FPGAs for Custom Computing Machines*, 1995.
[21] J. B. Peterson, R. B. O'Connor, and P. M. Athanas, "Scheduling and Partitioning ANSI-C Programs onto Multi-FPGA CCM Architectures," in *Fourth IEEE Workshop on FPGAs for Custom Computing Machines*, 1996.
[22] M. Figueiredo and C. Gloster, "Implementation of a probabilistic neural network for multi-spectral image classification on an fpga-based custom computing machine," *Proceedings of the 5th Brazilian Symposium on Neural Networks*, December 1998.
[23] D. L. Verbyl, *Satellite remote sensing of natural resources*, Lewis Publishers, Lewis Publishers, 1995.
[24] M. Birmingham S. R. Chettri, R. F. Cromp, "Design of neural networks for classification of remotely sensed imagery," *Telematics and Informatics*, vol. 9, pp. 145–156, 1992.
[25] S. R. Chettri and R. F. Cromp, "Probabilistic neural network architecture for high-speed classification of remotely sensed imagery," *Telematics and Informatics*, vol. 10, pp. 187–198, 1993.
[26] Giga Operations, *Giga Operations Spectrum Reconfigurable Computing Platform Documentation*, Giga Operation Corporation, 1995.

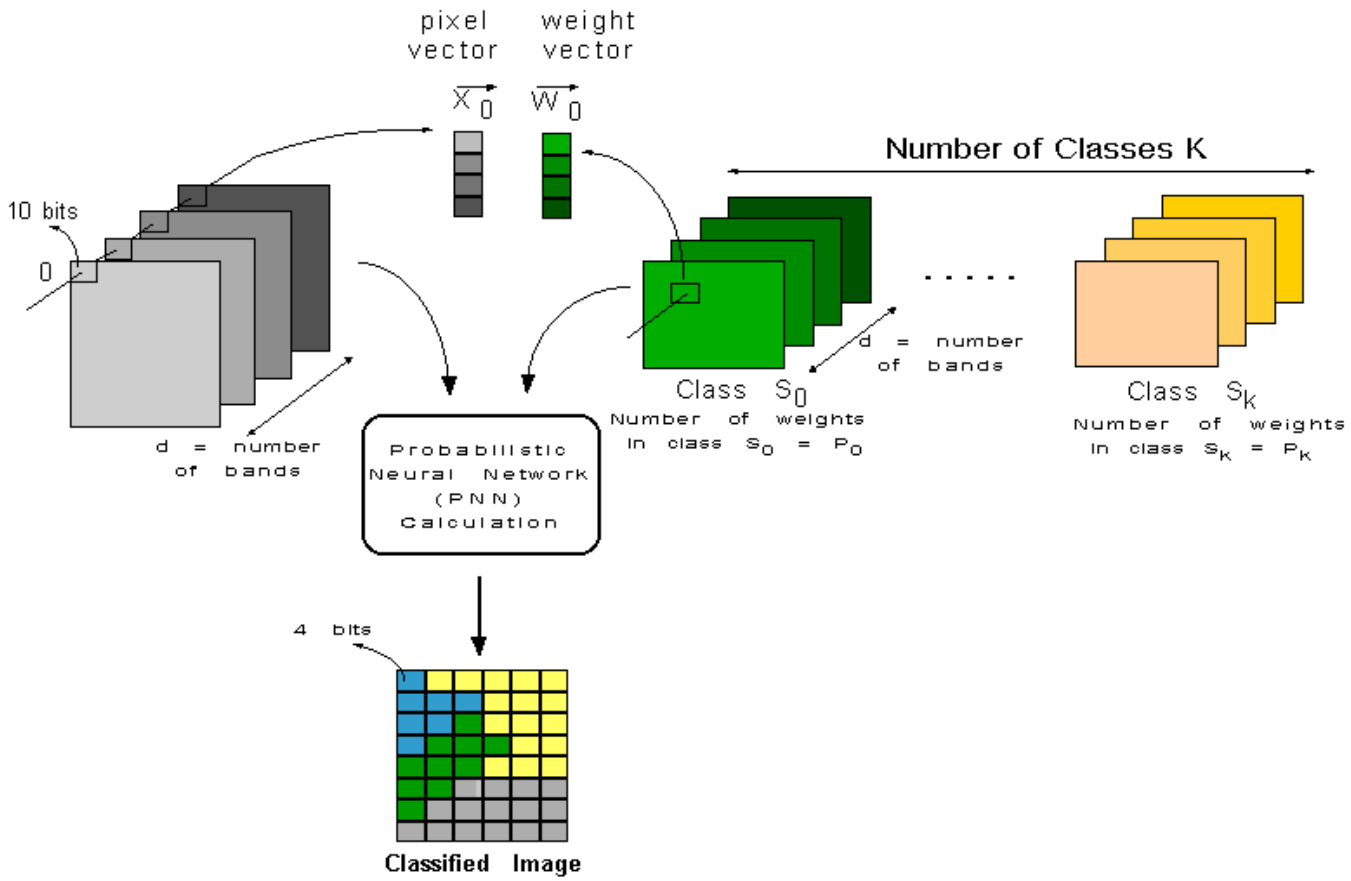

Fig. 1. PNN Image Classifier.

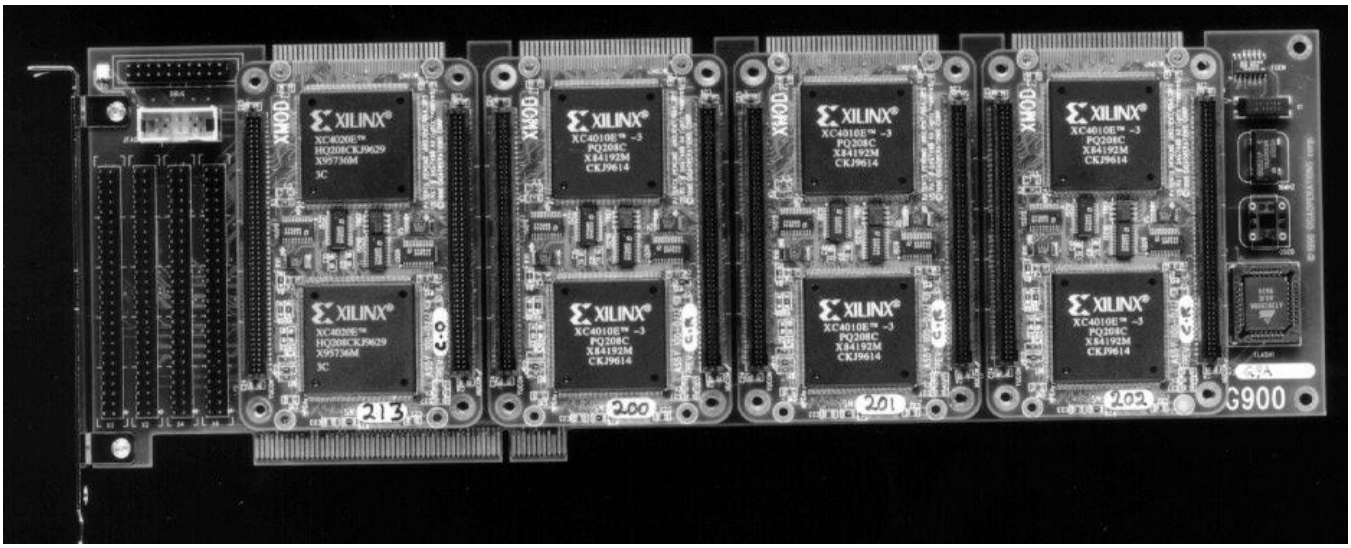

Fig. 2. FPGA board connected to a PCI bus slot.

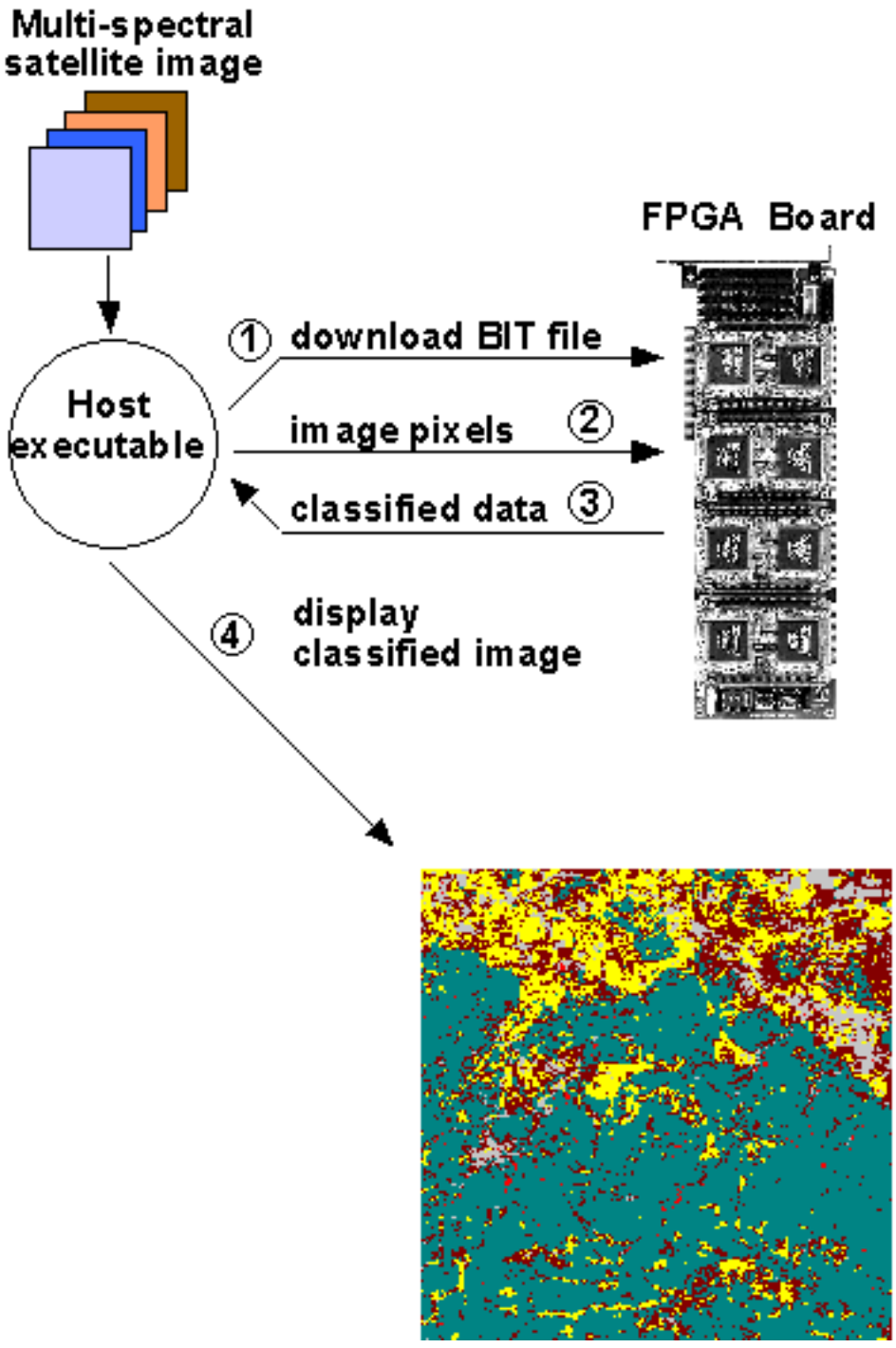

Fig. 3. Multispectral image classification using an FPGA coprocessor.

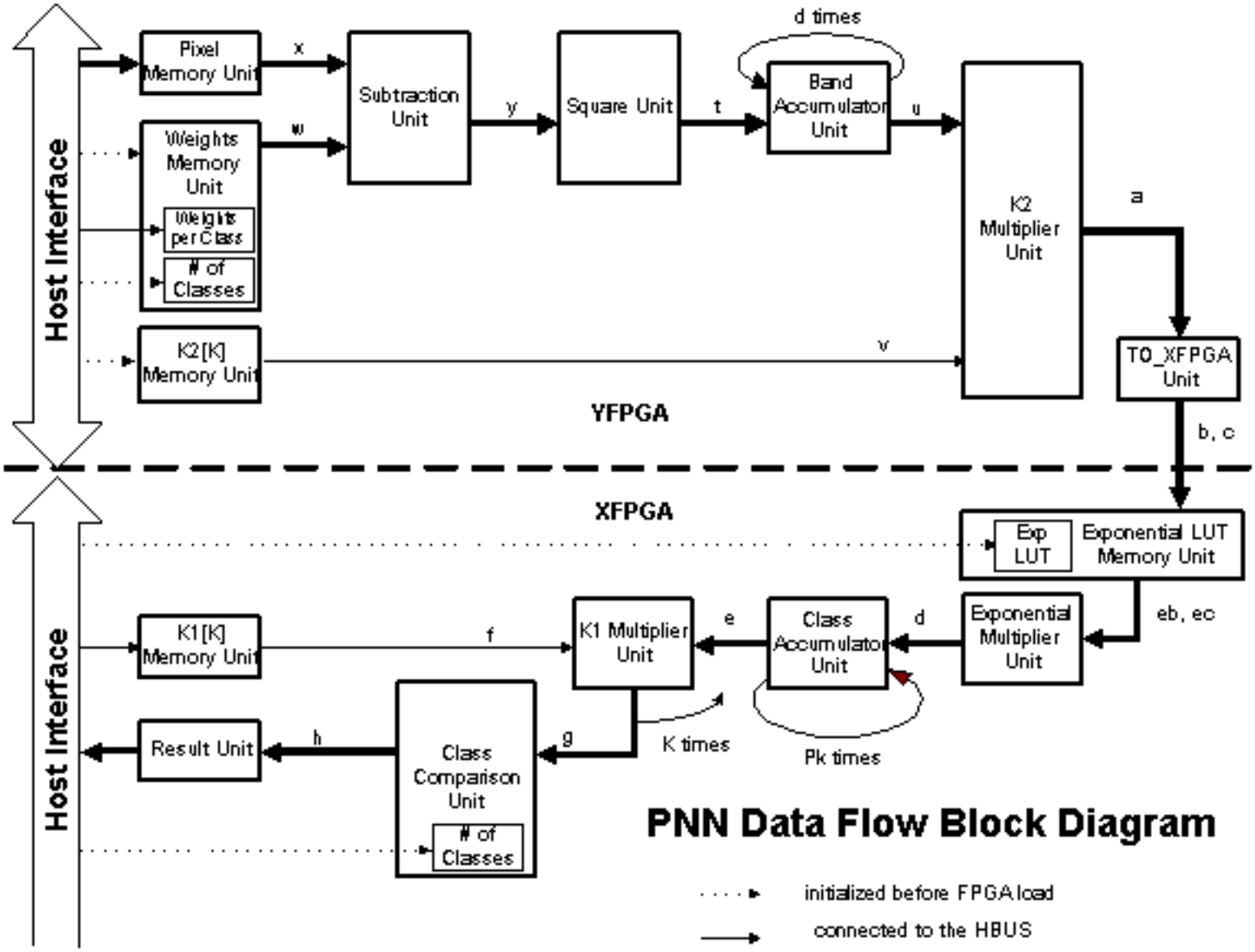

Fig. 4. PNN Classifier Data Unit.

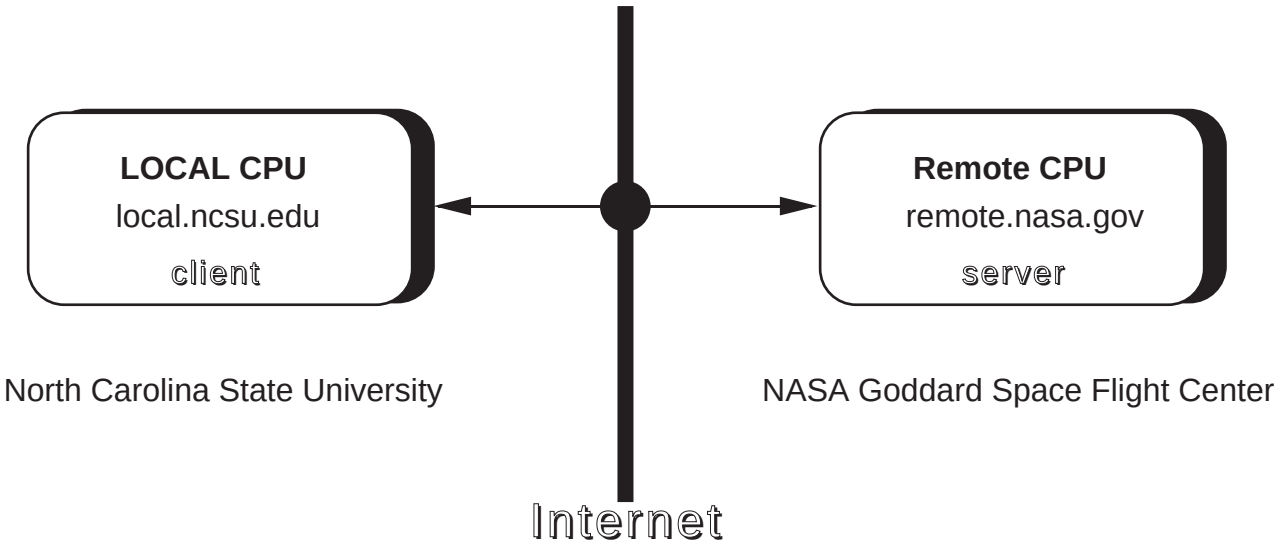

Fig. 5. Algorithm Execution on a Remote Adaptive Computer.

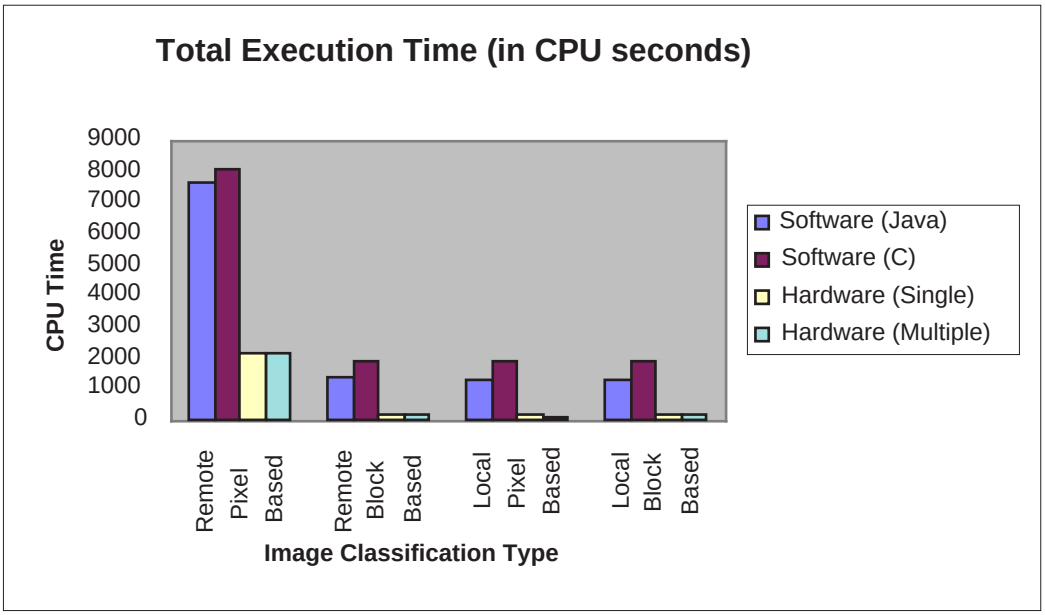

Fig. 6. Local and Remote Image Classification Execution Time.

TABLE I

Remote Image Classification Timing Report (in CPU seconds).

	Pixel Based		Block Based	
Description	**Avg Row**	**Total**	**Avg Row**	**Total**
Software (Java)	14.83	7597.79	2.65	1358.91
Software (C)	15.79	8087.24	3.61	1847.33
Hardware (Single)	4.16	2128.38	0.35	178.217
Hardware (Multiple)	4.25	2156.02	0.35	180.15

TABLE II

Local Image Classification Timing Report (in CPU seconds).

	Pixel Based		Block Based	
Description	**Avg Row**	**Total**	**Avg Row**	**Total**
Software (Java)	2.57	1317.31	2.56	1309.65
Software (C)	3.57	1871.36	3.69	1889.63
Hardware (Single)	0.27	141.10	0.28	143.05
Hardware (Multiple)	0.14	77.45	0.28	142.57

Marco Figueiredo is an R&D Engineer working in the investigation and application of reconfigurable computing at NASA Goddard Space Flight Center. He has a B.S. in Electrical Engineering from the Universidade Federal de Minas Gerais - Brasil (1988), and a masters in Computer Engineering from Loyola College in Maryland, USA (1991).

Clay Gloster, Jr. is currently an Associate Professor in the Department of Electrical & Computer Engineering at North Carolina State University. He received the B.S. and M.S. degrees in Electrical Engineering from North Carolina A&T State University and the Ph.D. degree in Computer Engineering from North Carolina State University. He also has been employed with IBM, the Department of Defense, and the Microelectronics Center of North Carolina. Current research focuses on the identification of potential applications and the development of automated tools that assist scientists/engineers in mapping these applications onto reconfigurable computing resources. He is also actively conducting research in the area of technology based curriculum development and distance education. Dr. Gloster is a member of Eta Kappa Nu, Tau Beta Pi, ACM, an is a registered professional engineer.

Mark Stephens is a Computer Engineer working for the NASA Goddard Space Flight Center. He has a B.S. in Applied Math, Theoretical Physics and Fluid Dynamics from Tulane University (1976). He went on to get a Master of Science in Atmospheric Science at Colorado State University (1979).

Corey Graves received his B.S. degree in Electrical Engineering from North Carolina State University in 1993 and his M.S. degree in Electrical Engineering from North Carolina State A&T State University in 1994. He is currently completing his Ph.D. Work in Computer Engineering at North Carolina State University. His current research interests include Run-time Reconfigurable Computing, Applications of Wavelets, Adaptive DSP algorithms, and Architectures for DSP. During his years a student, Corey has worked with many industrial and government entities including IBM, AT&T, Sandia National Labs, and Oak Ridge National Labs. He is currently a member of the IEEE Signal Processing Society.

Mouna Nakkar received her Bachelor's degree in Electrical Engineering from the University of North Carolina at Charlotte, NC in 1991, her Master's from the same university in 1993, and her Ph.D. degree in 1999 from North Carolina State University in Raleigh, NC. She joined Motorola Inc. in July of 1999 and is currently working on microprocessor research and development. Her research interests include reconfigurable computing, embedded processors, FPGA architecture, low power FPGAs, high speed CMOS design, Multi-Chip-Module Packaging, 3-Dimensional MCM packaging, and Optical Interconnects for high speed VLSI packaging. She worked on a microprocessor design during a co-op in 1995 at Ross Technologies, Austin TX. She also served as a teaching assistant for several electrical engineering courses.